Apellidos: **Nombre:** **Matrícula:**

Examen de Programación Concurrente - Clave a

Junio 2009

Departamento de Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este examen mezcla preguntas tipo test y preguntas de respuesta corta. Consta de **8 preguntas** en **6 páginas**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora y media**. El examen debe contestarse en las **mismas hojas**. No olvidéis rellenar **apellidos, nombre y número de matricula** en cada hoja.

Sólo hay una respuesta válida por pregunta tipo test. Toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas en la misma.

La solución al examen se proporcionará antes de la revisión. Las calificaciones se darán a conocer el **14 de julio**. La revisión del examen tendrá lugar el **16 de julio**.

Cuestionario

(1 punto) 1. El siguiente procedimiento a ser ejecutado simultaneamente por varios procesos.

```
procedure MCD (X, Y : in Natural;
               MCD  : out Natural) is
   B : Natural := Y;
begin
   MCD := X;
   while B > 0 loop
      if MCD > B then
         MCD := MCD - B;
      else
         B := B - MCD;
      end if;
   end loop;
end MCD;
```

Se pide marcar la afirmación correcta.

(a) ☐ Siempre es necesario asegurar exclusión mutua en el acceso a la variable `B`.

(b) ☐ Siempre es necesario asegurar exclusión mutua en el acceso a los parámetros `X` e `Y`.

(c) ☐ Siempre es necesario asegurar exclusión mutua en el acceso al parámetro `MCD`.

(d) ☐ Ninguna de las otras respuestas es correcta.

(1 punto) 2. Cuando el siguiente programa haya terminado se espera que haya impreso una linea con `110` ó `50`:

```
X : Integer := 100;

task P1;
task P2;

task body P1 is
begin
   X := X + 10;
end P1;
```

```
task body P2 is
begin
   if (X > 100) then
      Put_Line (Integer'Image (X));
   else
      Put_Line (Integer'Image (X - 50));
   end if;
end P2;
```

Se pide marcar la respuesta correcta asumiendo que la lectura y asignación de variables son acciones atómicas:

(a) ☐ El programa es correcto.

(b) ☐ El programa es incorrecto.

(1 punto) 3. El siguiente tipo de tareas `P` implementa un protocolo de acceso a una sección crítica.

```
task type P (I : PID);

task body P is
begin
   loop
      S;
      while Turno /= I loop null; end loop;
      Seccion_Critica;
      Turno := Turno + 1;
   end loop;
end P;
```

Debe asumirse que la sentencia `Turno := Turno + 1` es atómica y equivalente a
if `Turno = MAX_TASKS` **then** `Turno := 1;` **else** `Turno := Turno + 1;` **end if**.

Dado un programa concurrente con `MAX_TASKS` tareas de tipo `P` compartiendo una variable `Turno` inicializada a `PID'First` y cada una de ellas con un índice `I` distinto.

Se pide marcar la afirmación correcta.

(a) ☐ Asumiendo la terminación de `S` y `Seccion_Critica`, el programa no cumple la propiedad de exclusión mutua en `Seccion_Critica`.

(b) ☐ Asumiendo la terminación de `S` y `Seccion_Critica`, el programa no cumple la propiedad de ausencia de interbloqueo.

(c) ☐ Asumiendo la terminación de `S` y `Seccion_Critica`, el programa no cumple la propiedad de ausencia de inanición.

(d) ☐ Ninguna de las otras respuestas es correcta.

(1 punto) 4. A continuación se muestra un diseño de un sistema concurrente con una especificación formal de un recurso compartido (no se muestra el interfaz pero no contiene otras operaciones que las especificadas siendo el segundo argumento de las operaciones de tipo $\mathbb{Z}$) y tres tareas `T1`, `T2` y `T3` que comparten dicho recurso (variable `N : Notificacion`).

TIPO: $Notificacion = \mathbb{Z}$
INICIAL(n)**:** $n = 0$

CPRE: Cierto

Notificar(n,x)
POST: $n^{sal} = x$

CPRE: $n \neq x$

Sincronizar(n,x)
POST: $n^{sal} = n^{ent}$

```
task body T1 is
begin
   Notificar (N, 1);
end T1;

task body T2 is
begin
   Sincronizar (N, 0);
   Put (Integer'Image (0));
end T2;

task body T3 is
begin
   Sincronizar (N, 1);
   Put (Integer'Image (1));
end T3;
```

Se pide marcar la afirmación correcta suponiendo que las operaciones `Put` son atómicas.

(a) ☐ En ningún caso es posible que las tres tareas terminen.

(b) ☐ "`01`" es una salida posible del programa.

(c) ☐ "`01`" no es una salida posible del programa.

(d) ☐ "`10`" no es una salida posible del programa.

(1 punto) 5. Dada la siguiente implementación del recurso del problema 4[1]:

```
type Info_Sinc is record
   X : Integer;
   C : CB.Channel;
end record;

package Colas_Sinc is
   new Colas (Info_Sinc);
use Colas_Sinc;

procedure Notificar
  (N : in out Notificador;
   X : in     Integer) is
begin
   N.Notificar (X);
end Notificar;

procedure Sincronizar
  (N : in out Notificador;
   X : in     Integer) is
   C : CB.Channel;
   B : Boolean;
begin
   CB.Create (C);
   N.Sincronizar (X, C);
   CB.Receive (C, B);
   CB.Destroy (C);
end Sincronizar;
```

```
task body Notificador_RV is
  N : Integer := 0;
  N_Por_Sinc : Natural := 0;
  Por_Sinc : Cola;
begin
  Crear_Vacia (Por_Sinc);
  loop
    select
      when True =>
        accept Sincronizar
          (X : in Integer;
           C : in out CB.Channel) do
          Insertar (Por_Sinc, (X, C));
          N_Por_Sinc := N_Por_Sinc + 1;
        end Sincronizar;
    or
      when True =>
        accept Notificar
          (X : in Integer) do
          N := X;
        end Notificar;
    end select;
    declare
       Por_Atender : Natural := N_Por_Sinc;
       Pet : Info_Sinc;
    begin
       while Por_Atender > 0 loop
          Primero (Por_Sinc, Pet);
          Borrar (Por_Sinc);
          Por_Atender := Por_Atender - 1;
          if N /= Pet.X then
             CB.Send (Pet.C, True);
             N_Por_Sinc := N_Por_Sinc - 1;
          else
             Insertar (Por_Sinc, Pet);
          end if;
       end loop;
    end;
  end loop;
end Notificador_RV;
```

Se pide marcar la afirmación correcta.

(a) ☐ Es una implementación correcta del recurso compartido a pesar de que sólo se realiza una pasada por los elementos de la cola `Por_Sinc`.

(b) ☐ Pueden desbloquearse procesos que no deben, es decir, puede producirse una violación de una condición de sincronización.

(c) ☐ Pueden quedar bloqueados procesos que deben ser atendidos al cumplirse su condición de sincronización.

(d) ☐ Ninguna de las otras respuestas es correcta.

[1] **`declare`** `D` **`begin`** `S` **`end;`** es una sentencia de Ada que permite declarar variables (`D`) para un bloque de sentencias (`S`) y que provoca la ejecución de dichas sentencias.

(1 punto) 6. Se pretende modificar el esquema de código utilizado para implementar recursos compartidos mediante *rendez-vous* y paso de mensajes para utilizar sólo *rendez-vous*. La idea es la siguiente: por cada operación del recurso en la que fuera necesario el uso de un canal, se ha substituido dicho uso por una nueva entrada en el servidor. Dicha entrada es invocada por el cliente tras invocar la entrada principal asociada a dicha operación. El servidor no aceptará la entrada aplazada hasta el momento en el que sea posible. Veamos el esquema de código:

```
procedure Op (R : in out Recurso;
              X : in     T;
              Y :    out T) is
begin
   R.S.Op (X);
   R.S.Op_Aplazada (Y);
end Op;

task body Servidor is
   --  ...

   X : T := 0;
   R : T;
begin
   loop
      select
         when True =>
            accept Op (X : T) do
               Almacenar (X);
            end Op;
      or --  ...

      end select;
      --  Atención a tareas bloqueadas
      while Alguna_CPre_Cierta loop
         Recuperar (X);
         if CPre_De_Op (X) then
            Realizar_Op (X, R);
            Borrar (X);
            accept Op_Aplazada
                      (Y : out T) do
               Y := R;
            end Op_Aplazada;
         end if;
      end loop;
   end loop;
end Servidor;
```

El resto de la metodología mostrada en el curso (el bucle de desbloqueo, los parámetros de entrada salida, la comprobación de la condición de sincronización, la realización de la operación en si, el almacenamiento y recuperación de los datos, etc.) se aplica sin cambios.

Se pide marcar la afirmación correcta.

(a) ☐ La linea **accept** `Op_Aplazada` en el servidor es un error de sintaxis en Ada 95.

(b) ☐ El funcionamiento del programa es el mismo que con el canal si solo hay un cliente que invoque la operación `Op`.

(c) ☐ No se desbloquea nunca a un cliente para el que no se cumple la condición de sincronización.

(d) ☐ El esquema presentado provoca interbloqueo.

(2 puntos) 7. Una de las tareas más delicadas en el diseño de un programa concurrente es la identificación de procesos. Enumerar las principales directrices que pueden ayudarnos a realizar un buen descubrimiento de procesos.

Directrices para identificar procesos

(2 puntos) 8. En un recurso compartido la condición de sincronización de una operación depende de un parámetro de entrada. Describe las ventajas y desventajas de implementar el control de dicha condición con una familia de *entries* indexada por el número máximo de procesos del sistema:

Ventajas

Desventajas

NO USAR ESTE ESPACIO